

I²C Servo Interface

control eight servos using the I²C bus

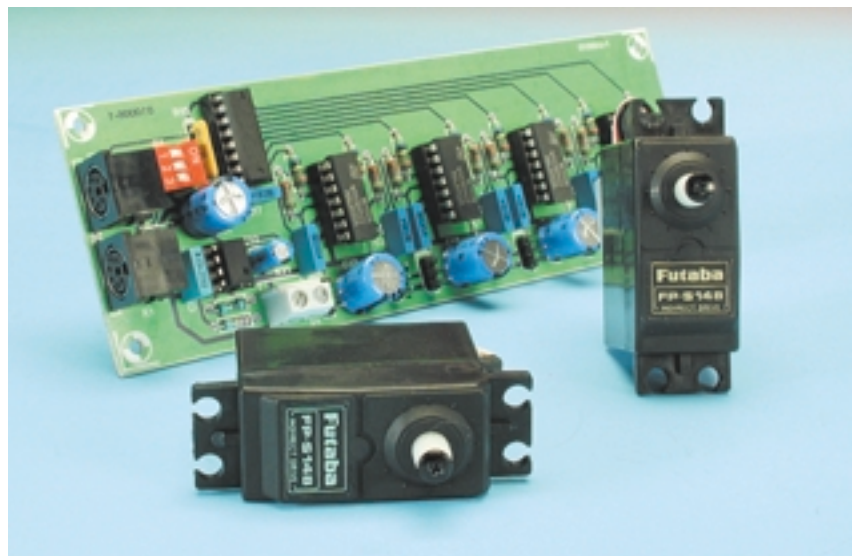
Design by C. Fuchs

Model servos have improved greatly in the past few years and now offer greater torque and speed, so much so that they are increasingly being used for other applications. The circuit described here controls up to eight servos using the well-established I²C bus. Additional cards enable expansion up to 64 servos. Software running under Win95/98 demonstrates the features of the circuit and allows simple programming on a PC using Visual Basic.

Generating an accurate Pulse Width Modulated (PWM) signal necessary to control a model servo is not an easy job for any computer particularly so if its running a multi-tasking operating system like Windows. Needless to say the problem gets worse when we add more servos and need to generate more control waveforms. In this situation the application of a little hardware often works wonders. The solution presented here uses a simple card driving eight servos controlled from an I²C bus on a parallel PC interface like the one presented in the March 1999 PC Topics Supplement of *Elektor Electronics*. The card allows eight servos to be controlled independently via commands sent over the I²C bus. Address selector switches on the card enable eight of these cards to be connected in parallel to the same I²C bus thereby providing a maximum of 64 servos to be controlled from the same I²C bus.

Circuit overview

An outline of the servo controller circuit is shown in **Figure 1**. The NE555 timer (IC6) is configured as an astable multivibrator, generating a pulse every 20 ms to trigger the eight monostable multivibrators (monoflops) formed by the four 556 dual timers (IC2-IC5). The pulse width output signal from these ICs will control the movement of each servo connected to that channel. A 1-ms wide pulse will move the servo arm to one end of its travel while a 2-ms pulse will move it to the



other end (usually 90 degrees of movement). The control is proportional so that intermediate pulse widths will move the arm to the corresponding intermediate position. The control voltage for each of the eight monoflops is produced by an 8-channel DAC type TDA8444, which interfaces to an I²C PC port and can be controlled using software routines running in Visual basic on the PC.

High torque/speed servos can generate appreciable levels of electrical interference and despite care-

ful PCB layout and supply decoupling this has led to occasional malfunction of the TDA8444. If you plan to use many servos it is advisable to power the servos independently from an external supply and use the +5 V derived from the I²C connector for the circuitry on this interface card only.

Circuit details

IC1 is the main component of the circuit diagram shown in **Figure 2** it is an 8-channel Digital to Analogue

Converter (DAC). The voltage level of the eight analogue outputs can be adjusted with a 6-bit resolution, giving 2^6 or 64 steps. Information to the DAC is supplied from the clock and data lines (SCL and SDA) of the I²C interface over the parallel wired mini DIN connectors K1 and K2. These connectors also allow the interface card and servos to be powered from the I²C bus. Details of the command structure are given later.

The address of the DAC on the I²C bus can be set up on DIP switch S1. This corresponds to bits 2, 3 and 4 of the address (0100_A2_A1_A0_0). The address switch allows up to eight interface cards to be controlled over the same I²C bus.

The VMAX (Pin 2) input of the DAC can be used to reduce its output voltage range if it is connecting to lower voltage circuitry. In this

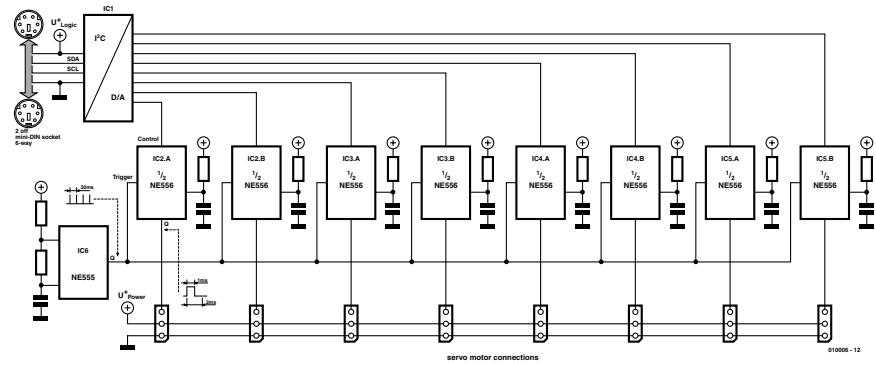


Figure 1. Servo interface block diagram.

application we are using 5 V throughout so VMAX can be left open-circuit. The output voltage of each channel (pins 9-16) of the DAC are connected via resistors R3-R10 to the control inputs (CV) of the dual timer IC's. The voltage level will be

approximately in the range of 0.26 V to 3.12 V.

The NE556 dual timers are connected in the classic monoflop configuration. The timing components $R = 18.7 \text{ k}\Omega$, $C = 100 \text{ nF}$ together with a control voltage input of 0.26 V on the $6.04 \text{ k}\Omega$ series resistor will give an output pulse width of 1.0 ms. Increasing the con-

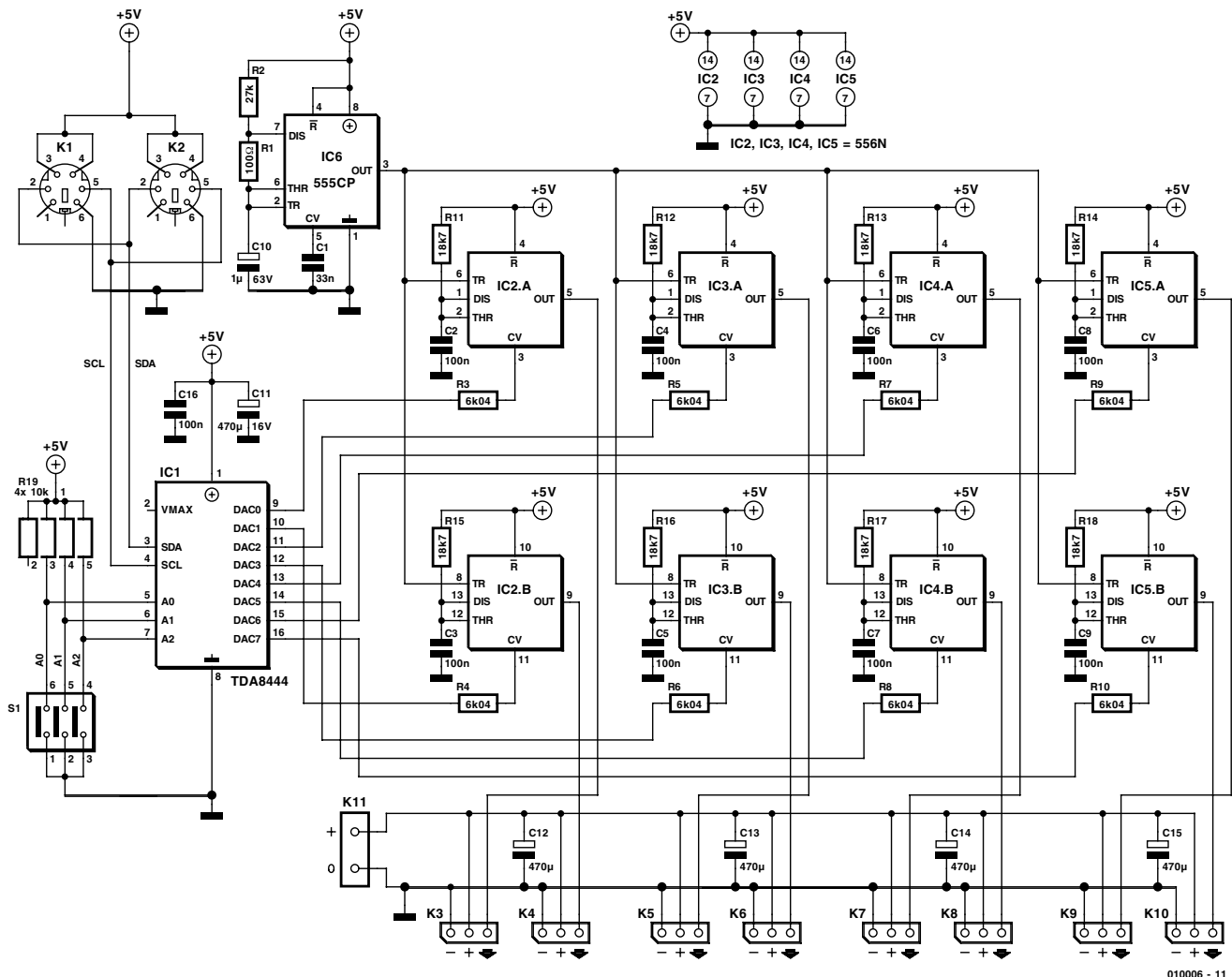


Figure 2. Servo interface circuit diagram.

trol voltage up to its maximum of 3.12 V will produce a 2.0 ms pulse. These two pulse widths represent the maximum and minimum travel of the servo and should produce a 90 degree travel of the actuating arm. In practice, component tolerances may introduce an offset or reduce the travel so it may be necessary to alter component values to achieve exactly 90 degrees if this is important for your application. The 1 ms to 2 ms pulse width signal to each servo must be repeated approximately every 20 ms. A free running astable multivibrator formed by IC6 provides this signal by generating a short 1 ms pulse every 20 ms to simultaneously trigger each of the eight monoflops. The output pulses are then taken to the three pin output connectors on the PCB along with the supply rails to drive the servos.

Different servo manufacturers in the past have always tended to use their own pin designations for the servo connectors and if you have some of these older servos the pin configurations shown in **Figure 3** should help. Be careful here, an incorrectly wired servo will be an expensive mistake. Thankfully, the situation in the UK is improving and most modern servos share a common pin configuration and are interchangeable. As mentioned earlier, servos can generate a high level of elec-

trical noise so the PCB for this circuit is laid out with wide supply rails to the servo connectors and large supply decoupling capacitors to reduce this interference.

Construction

The PCB layout is shown in **Figure 4**. Building the card should begin by fitting and soldering the more robust components such as the wire links (9 in all), the IC sockets, connectors and pin strips to the PCB. Next, fit the passive components. Don't forget that the resistor array R19 must be fitted the correct way round. Before the IC's are fitted into their sockets, it is a good idea to connect the card to ensure that the +5 V supply is available on all the power supply pins of the IC's. Only if everything looks OK can the IC's be fitted and the card connected to the I²C interface. If many servos are connected it is important to use an external +5 V power supply and not rely on the I²C bus supply.

After power up, check that the outputs of the eight DAC channels

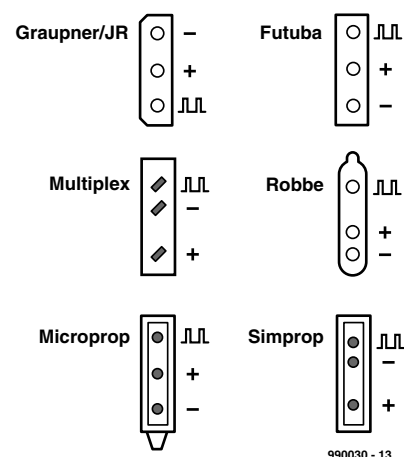
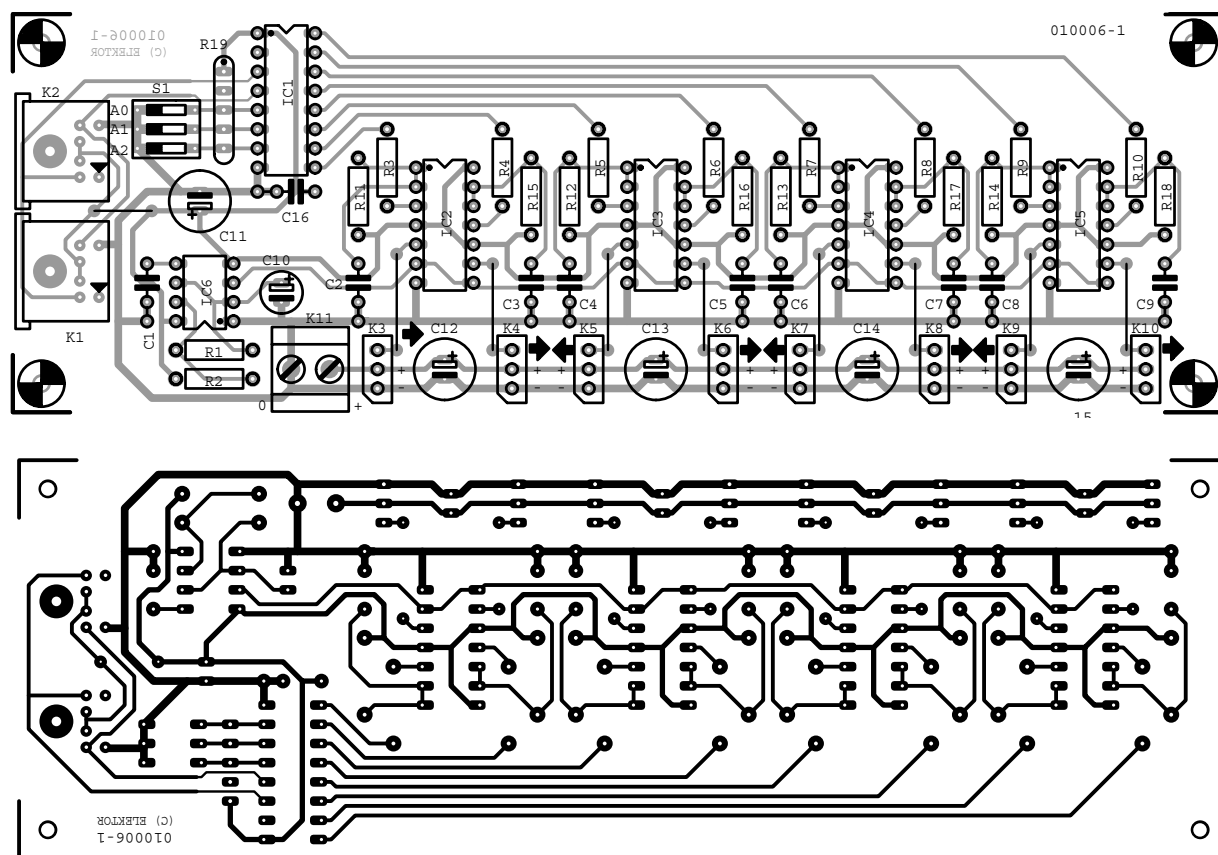


Figure 3. Pinouts of some popular servos.

are at 0 V (0.26 V maximum). Connect an oscilloscope to the output of IC6 (pin 3) and look for the short 1 ms pulse repeated every 20 ms. When a servo is plugged into the card, its output arm should drive to an end position. If all functions are satisfactory then the circuit up to the TDA8444 should be OK.



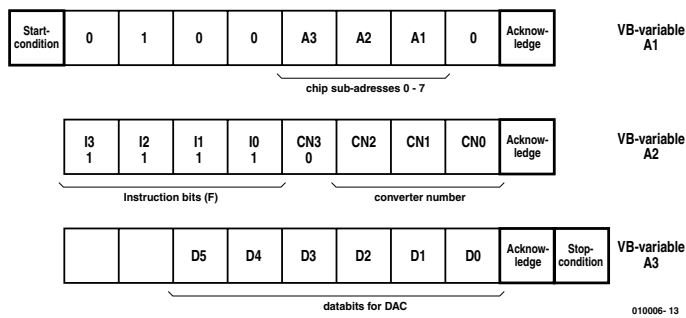


Figure 5. Format for programming the TDA8444.

The Software

The demonstration software **i2cservo.exe** is a 16-bit program and only runs under Win95/98 and not under NT. The file **iic.dll** contains software to interface to the PC hardware and must be placed in the same folder as the application or in the *system* folder in Windows. Visual Basic 3 was used to write the software. Options in the menu list allow selection of the parallel port LPT1 (378hex) or LPT2 (278hex) and also the subaddress which is decoded on the card by DIP switch S1. A 7 will talk to an interface board which has the all three switches connected to +5 V while a 0 will talk to one which

has all three connected to ground. The 8 scroll bars correspond to the eight DAC output channels and therefore the servos. Clicking and dragging on one with the mouse will cause the corresponding servo to move. This simple executable demonstration programme written in Visual Basic 3 together with the source code and the PCB layout files are available as a free download from the *Elektor Electronics* website. This program is a good starting point for developers and is relatively easy to modify and expand.

When the program first starts the default chip sub-address is set to 7 (all of the address pins are switched to +5 V)

```
ChipAddress = Val(Text1.Text)
```

and the port used is set to LPT2.

```
Lpt = Lpt2
```

LPT1 and LPT2 are defined in **Modul1.bas** and have the values &H378 and &H278 respectively. Both or the settings (7 and LPT2) can of course be redefined.

```
(SubText1_Change,  
SubParallel1/2_Click).
```

Next comes the check to determine if the I²C interface is connected.

```
ErrorMsg=TestConnected(LPT)
```

The corresponding failure message (if any) will be output. During operation each movement of a scroll bar

```
SubVScrollX_Change()
```

is read with X = 0 to 7 corresponding to scroll bars 1 to 8. The value of the setting is read (in the range 0 to 63) and assigned to the variable A3.

```
DacValue = VScrollX.Value
```

The maximum value MAX of the vertical scroll bars are set at 63 so there is no need to convert this value. The three least significant bits of A2 contain the code for the DAC number (0 to 7) which corresponds to that scroll bar. Bit 4 is always set to a 0 and all of the higher four bits are always set to a 1 (= 240 = Instruction F: sending data without an address will always write to the same DAC). The variable A1 contains bits 2,3 and 4 which defines the chip sub-address. The upper 4 bits contain the fixed chip address 0100 (64_d).

The three variables A1, A2 and A3 shown in **Figure 5** represent the data telegram sent over the I²C bus to the TDA8444 every time a scroll bar is moved. The routines required for this are provided in **iic.dll**. These functions are defined under GLOBAL. Sending information over the I²C bus begins with

```
ErrorMsg = StartCon( LPT)
```

To send the I²C-Start condition, and then

```
ErrorMsg=Transmit(LPT,A1..A3)
```

is used to send the three bytes. After sending the I²C bus will be released again by using the stop command

```
ErrorMsg = StopCon(LPT)
```

Any errors detected are assigned to five error variables **Er1** to **Er5** and displayed in the lower status window. See the box 'Error codes' for a list of the possible error messages. For further information on the TDA8444 a data sheet can be downloaded from the Philips Semiconductors website at http://www.semiconductors.com/acrobat/data_sheets/TDA8444_3.pdf

(010006-1)

COMPONENTS LIST

Resistors:

R1 = 100Ω
R2 = 27kΩ
R3-R10 = 6kΩ04, 1%
R11-R18 = 18kΩ7, 1%
R19 = 10kΩ 4-way SIL array

Capacitors:

C1 = 33nF
C2-C9,C16 = 100nF
C10 = 1μF 16V radial
C11-C15 = 470μF 16V radial

Semiconductors:

IC1 = TDA8444
IC2-IC5 = NE556
IC6 = NE555

Miscellaneous:

K1,K2 = 6-way mini-DIN socket
K3-K10 = 3-way pinheader
K11 = 2-way PCB terminal block, lead pitch 5mm
Disk, project software, order code **010006-11**

Error codes

- | | | |
|---|------------|--|
| 0 | Error 0000 | No failure detected |
| 1 | Error 0001 | The I ² C bus is not free. |
| 2 | Error 0002 | No acknowledge received. |
| 3 | Error 0003 | Unable to generate the stop condition. |
| 4 | Error 0004 | Voltage failure on the I ² C bus. |
| 5 | Error 0005 | The I ² C Interface is not connected. |
| 6 | Error 0006 | An undefined error has occurred. |